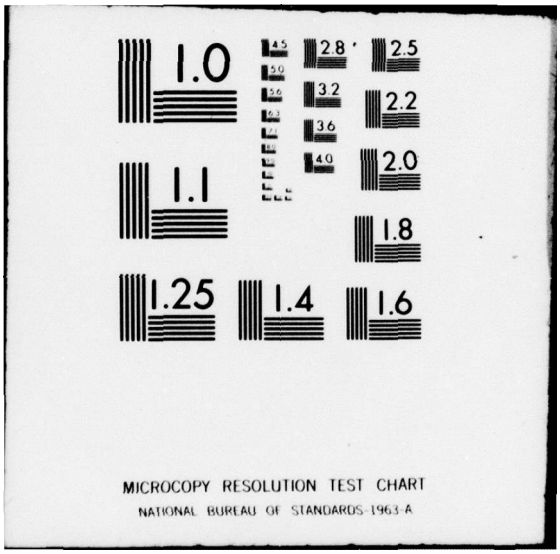
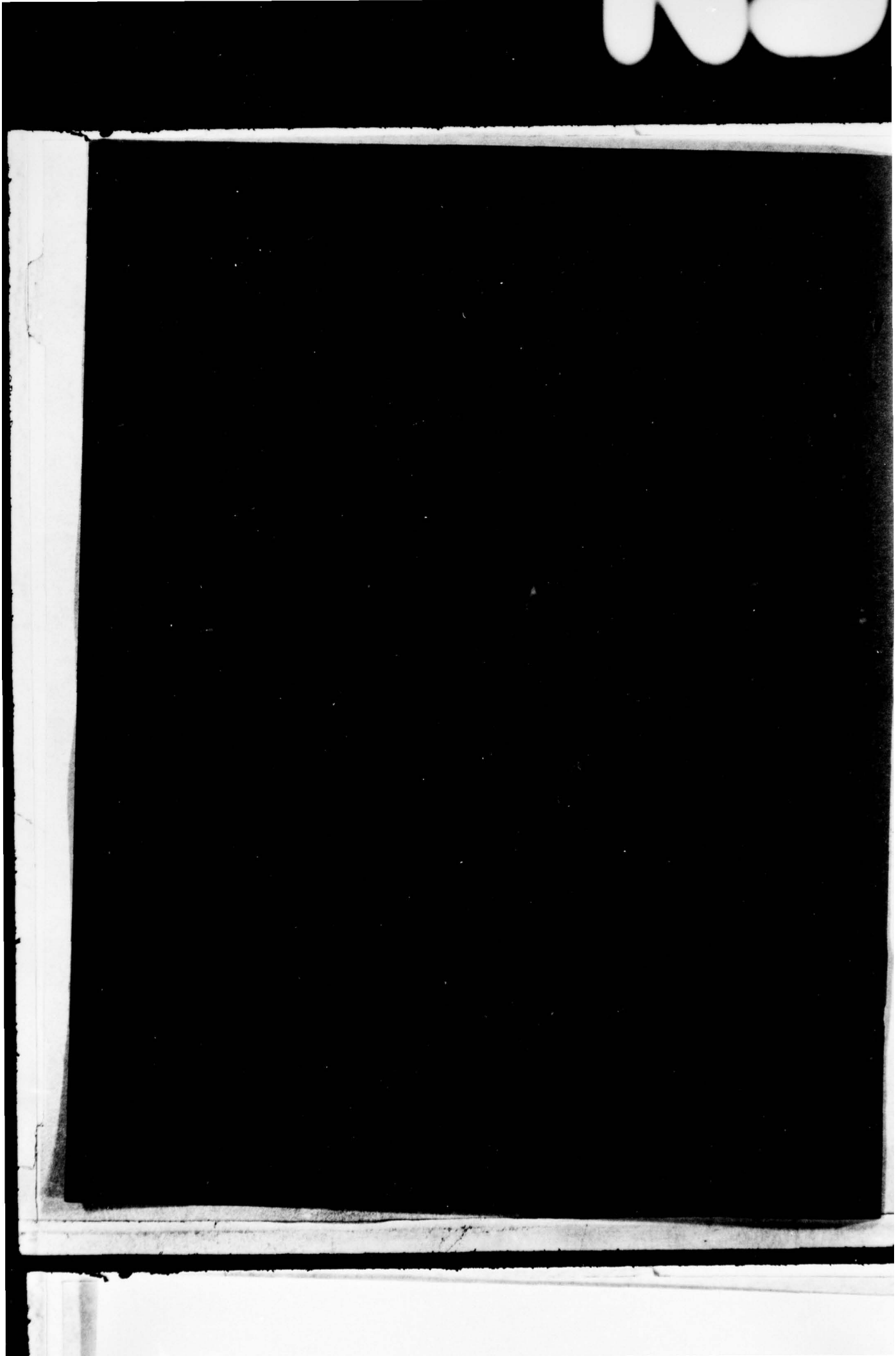
AD-A065 934 NAVAL SURFACE WEAPONS CENTER DAHLGREN LAB VA F/G 9/2
MICRO IMPLEMENTED LOADER--AN ADVANCED LOADER FACILITY FOR A MIC--ETC(U)
NOV 78 A P GASS
UNCLASSIFIED NSWC/DL/TR-3898 NL

END
DATE FILMED
5-79
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (14) NSWC/DL/TR-3898	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) MICRO IMPLEMENTED LOADER--AN ADVANCED LOADER FACILITY FOR A MICROPROGRAMMABLE COMPUTER,		5. TYPE OF REPORT & PERIOD COVERED (9) Final Rept.
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10) ALBAN P. GASS		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Surface Weapons Center (K74) Dahlgren, VA 22448		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Computer Program Support
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Surface Weapons Center (K74) Dahlgren, VA 22448		(11) 12. REPORT DATE November 1978 13. NUMBER OF PAGES 23
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) (12) 22p.		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution unlimited; approved for public release.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer, Loader, Linker, Binder, Object Code, Segments, Overlays, Linkage Editor, Operating System, Firmware Microprogrammed, SIMPL-Q, Emulation Aid System - EASY, Nanodata QM-1, Micro Implemented Loader (MIL)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The micro implemented loader (MIL) represents an advance in computer program loading technology for microprogrammable computer architecture. The MIL facility replaces, with a simpler approach, the complex system software traditionally associated with this function. It provides dynamic loading, linking, and delinking of computer modules and supports object library functions with a minimum of user and implementor effort. MIL takes advantage of special data structures and firmware to achieve its efficiency. → next page

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

79 03 16 018

391398

20. (Continued)

The loader was developed and implemented at NSWC on a Nanodata QM-1 computer in less than 200 micro locations.

FOREWORD

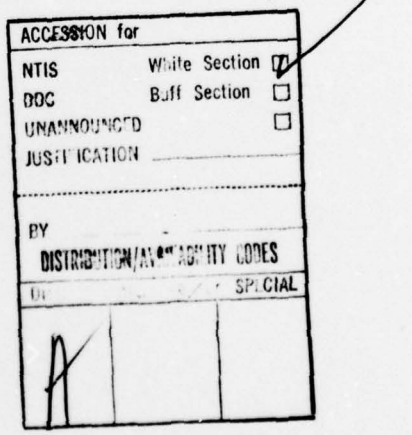
The micro implemented loader (MIL) was designed and implemented as a capability of an operating system called EASY (Emulation Aid System) for the Nanodata QM-1 microprogrammable computer at NSWC. The project was supported by the FBM Geoballistics Division and by the Computer Facilities Division.

This report was reviewed by Mr. Hermon W. Thombs, Head of the Programming Systems Branch, Computer Programming Division.

Released by:

Ralph A. Niemann

RALPH A. NIEMANN, Head
Strategic Systems Department



CONTENTS

INTRODUCTION

During the requirements definition phase of a project for the development of an operating system called EASY (Emulation Aid System),[1,2] the need for an effective and efficient program loading facility was noted. A loader was required that supported linking, libraries, overlays, and, if possible, dynamic allocation and delocation of code modules. There was a serious time and manpower constraint that made minimal implementation mandatory. Since this need became apparent early in the operating system project, it was possible to influence the design of the operating system and the underlying control structure of the machine architecture. The high-level language SIMPL[3,4] was chosen as the language for implementing system and user routines. Also, the implementor of the compiler for the SIMPL computer programming language was available to generate the object code in the structure specified.

A search of the literature revealed only meager information on loaders.[5-8] Very little information on the subject stating useful general principles with respect to microprogrammable computer loading was found.

The goal of the design then became to develop a loader facility that was flexible and powerful enough to support the end use in an effective and efficient manner but not to encumber the end user or implementor with unneeded complexity. Mr. Charles W. Flink II and Mr. John G. Perry, Jr., helped refine the design leading to implementation. The micro implemented loader (MIL) design was completed in May 1976 and, except for some minor enhancements, implemented by December 1976.

FUNCTIONAL DESCRIPTION

OVERVIEW

The MIL represents an advance in program loading technology for microprogrammable computer architectures. The effective use of data structures[9] and firmware accounts for the efficiency of the MIL. Its design supports all of the standard functions associated with traditional computer subsystems, which use complicated linkers, loaders, and library facilities requiring many thousands of instructions to complete their tasks. MIL accomplishes loading of code, linking of global data, binding of externals to entry points, and code relocation without the expected complexity. The more advanced loader functions such as libraries and bound object code modules (i.e., sometimes referred to as overlays or segments) are also supported. These bound object modules are dynamically loaded and freed.

Finally, there is the almost unique feature of delaying binding until the actual time of first execution of the instruction that references the external or global data. The Burroughs Corporation B7000/B6000 computer user loader has this characteristic.[10] This feature makes for a more efficient utilization of computer resources. It also saves on computer processor utilization and computer memory space, since only those references actually used in the particular run need be resolved and reside in main store.

Traditionally, loaders with this type of capability require a large amount of code to perform their function. The MIL system has no high-level code; all of the coding was done in microcode. The operating system or user program activates the loader indirectly by any instruction whose reference is unresolved or explicitly by two machine instructions (i.e., PSLOAD, PSFREE). The entire loader is implemented in microcode on a Nanodata QM-1 computer[11,12] in less than 200 words. True ease of implementation, reliability, and efficient operations can be obtained when system functions are reduced to a module of this size.

RELOCATION

One of the fundamental functions of any loader facility is the handling of relocation of machine addresses. A conventional approach by a loader facility might require all machine instruction modules to generate the code as if the module was going to execute at memory address zero. The loader would also require the generator of the machine module to supply the relocator with a mapping of what instruction addresses would have to be changed if the module was executed at some other address in the machine. This procedure is generally necessary since most conventional machines require machine instructions to contain *absolute memory address*. The computer needs to know exactly what location to reference. In traditional loader facilities this binding of instructions to memory address occurs at load time.

However, the EASY operating system resolves this problem by defining in microcode a machine where instruction addresses needing relocation are replaced with displacements between instruction location and the memory word referenced. In order for the firmware to compute the real computer memory address pointed to by the machine instructions, the microcode adds the displacement given in the machine instruction to the location of the instructions. This sum is the absolute memory address needed to be referenced. This form of addressing is sometimes referred to as relative addressing.

Since all code generated for this micro machine is done by one translator, the computation of this displacement is handled transparently to the end user. Implementation of this design feature eliminates a large part of the overhead performed by the loader.

LINKING

Another fundamental function of any loader is the linking together of separately generated load modules. It is very useful for a routine generated

at one time to reference another routine or data location generated at some other time. The traditional method is to modify the instructions address of all references to the external location with absolute address of the location before execution starts. This approach requires much overhead in table space and central processor time for externals possibly not used. This process is generally known as resolution of external references, binding, or linking.

The MIL design takes a different approach. No resolution of externals takes place until the actual instruction referencing the external is executed. The firmware is aware of a link list of entry points running through the machine code modules. The firmware makes use of this data structure to resolve the address dynamically. MIL may load a module into core to satisfy the external required.

DETAILED DESCRIPTION

THE OBJECT MODULE DATA STRUCTURE

A detailed description of the data structures of the object modules and their relation to other parts of the system is presented in this section. Each object/load module can consist of up to seven separate tables. The format and relation of each table are illustrated in Figure 1. The labels associated with the first word address of each table are given on the left of the figure. The tables are

MHDR*	Module Header Table
MID*	Module ID Table
EXTM	External Name Table
GLOBALS	Globals Table
ENTM*	Entry Point Name Table
TEXT	Text Table
ENDM*	End Module Table

* Required as part of any object module

It should be noted that, in this particular implementation, program space started at high memory addresses and grew towards low memory addresses. So the "top" or oldest module is found at the foot of the diagram, and the most newly loaded module or "bottom" is at the head of the diagram. Two pointers, Bottom of Program Space (BPS) and Top of Program Space (TPS), are also given in Figure 1. The function of TPS is to indicate the start of the program space, whereas BPS indicates the end of the currently used memory. As more space is needed, BPS is decremented.

The purpose and structure (Figures 2 through 8) of each table illustrated in Figure 1 are presented in the following pages. In the implementation decribed, the tables were reconstructed in 18-bit QM-1 computer words. However,

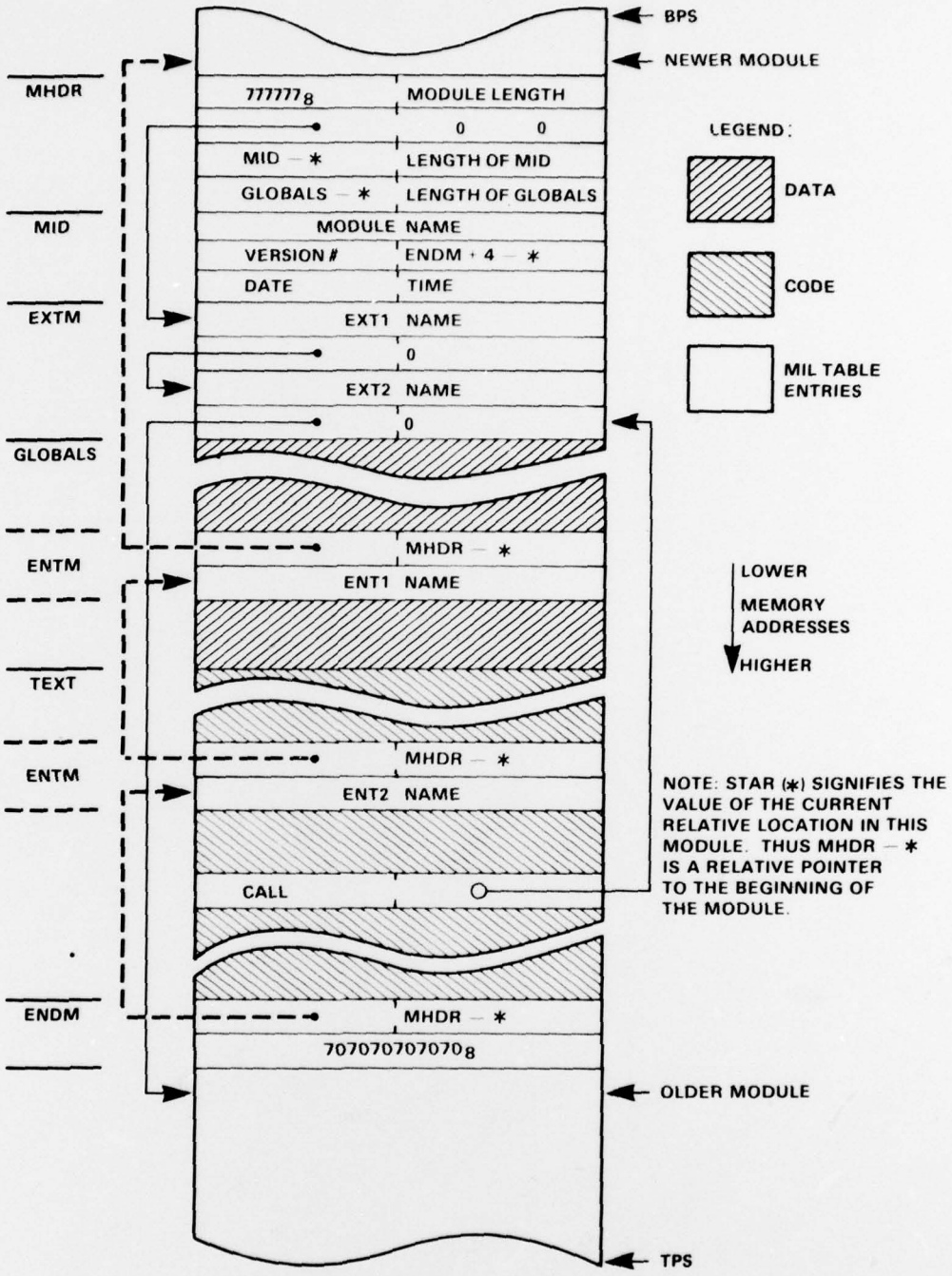


Figure 1. MIL Load Module with 2 Externals EXT1, EXT2 and 2 Entry Points ENT1, ENT2

the EASY user is only aware of the machine as a 36-bit machine. Transparent to the user two QM-1 words were used to make a SIMPL-Q word. However, 18-bit fields are efficiently implemented in this application so most of the fields are half EASY words.

Module Header Table (MHDR)

Purpose

1. To identify start of a legitimate module
2. To function as a dummy external link
3. To allow location of other tables within the module to be defined

MHDR			
	0	777777_8	LENGTH OF MODULE
	2	FIRST EXT POINTER	0
	4	MID POINTER	LENGTH OF MID
	6	GLOBALS POINTER	LENGTH OF GLOBALS

Figure 2. Module Header Table

Description of Entries

Word 0 - All sevens to flag start of module
- Used with word 1 to form a dummy external name

1 - Length of module, binary number
- A relative pointer to start of next higher module

2 - Relative pointer to first external item in module or, if none, to ENDM + 4 (i.e., the next MHDR)

3 - Zero

4 - Relative pointer to module ID table

5 - Length of module ID table

6 - Relative pointer to globals table

7 - Length of globals table

Note

If other tables are needed in the future, a relative pointer should be inserted at the end of the header table along with a word containing the length of the table.

Module ID Table (MID)

Purpose

To allow human identification of the module

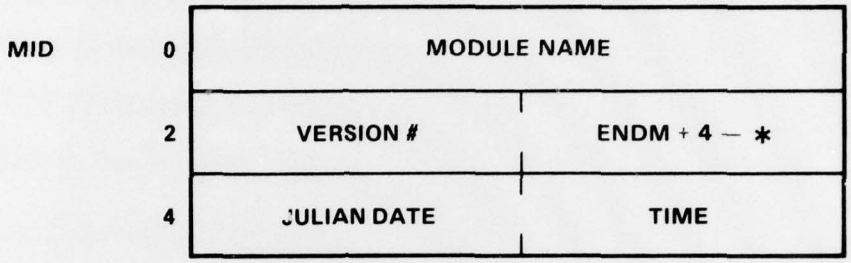


Figure 3. Module ID Table

Description of Entries

Word 0 and 1 used to contain name of module as defined in SIMPL-Q

2 - Version number of SIMPL-Q compiler being used to generate module (binary number)

3 - Relative pointer to start of next higher module in core (i.e., next MHDR)

4 - Julian date (binary number representing YYDDD)

5 - Time of compilation (binary integer representing HHMMSS)

Note

This table contains human identification information. For this table and all other loader tables, names are 1-6 characters represented with 6-bit ASCII characters. All names start with an alpha character (01-33 octal) and are left-justified within the word and zero-filled. If the name is used for external or entry data (as opposed to external or entry PROC/FUNC), then the leftmost bit is set to 1. This guarantees that external linkages can only be made to the correct type of links.

External Module Table (EXTM)

Purpose

To allow an external reference to be satisfied dynamically

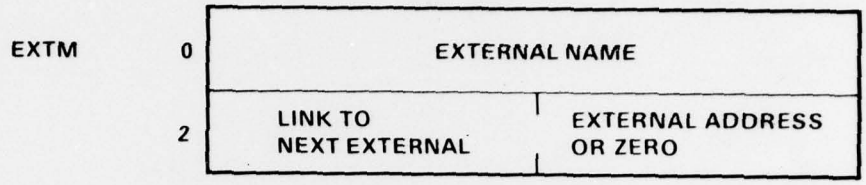


Figure 4. External Module Table

Description of Entries

This table consists of zero or more quadruples of words of the following format linked together.

Word 0 and 1 - Name of external, up to six 6-bit coded characters

2 - Relative pointer to next external quadruple

3 - Initially zero; when resolved, the relative address corresponding to the external name

Note

All references to this external within a module normally refer to the same quadruple.

In large modules where a reference to this table may be greater than 10000_8 locations, a second table for the same external name will be generated. Caution must be used in dynamically redefining such multiple external tables, since the table could be redefined and another table used in the call statement executed.

The names of all external data references are coded so a distinction between transfer (call) and a data reference can be achieved. Data references have the high-order bit of the first character set.

Globals Table

Purpose

To group all global variables together

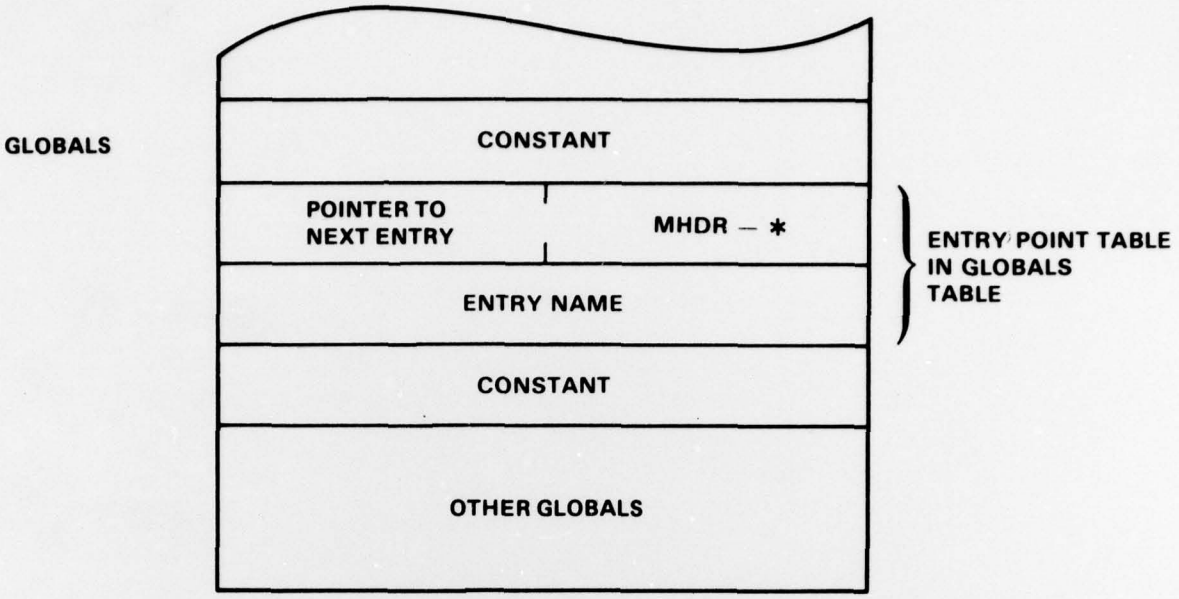


Figure 5. Globals Table Containing an Entry Point Table

Description:

The globals table consists of zero or more locations for global variables. If it is an entry global variable, then an entry table (with the global name coded as a data item) precedes the location of the global variable.

Entry Table (ENTM)

Purpose

To allow reference of data or PROC/FUNC from another module

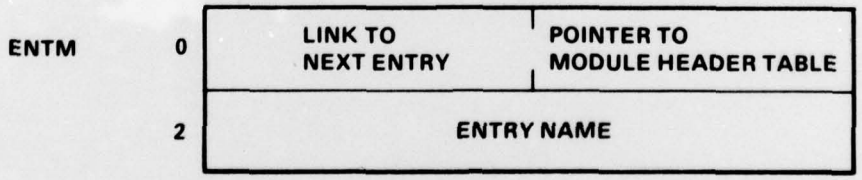


Figure 6. Entry Table

Description

An entry table may occur in one of two places--either in the globals table (as shown in Figure 5) for a data item or in the text table (EASY code) for ENTRY PROC or FUNC (Figure 6). In either case, the item (code or data) being referenced immediately follows the entry table. Regardless of the location of an entry point, the format is:

Word 0 - Relative pointer to the preceding entry point in the module; if none, then relative pointer to 2 words before module (i.e., ENDM + 2 of preceding module)

1 - Relative pointer to module header table

2 & 3 - Name of the global or entry point

Note

Data names for globals and PROC/FUNC must follow the convention discussed under the module ID table.

Text Table

Purpose

To provide an area for executable code

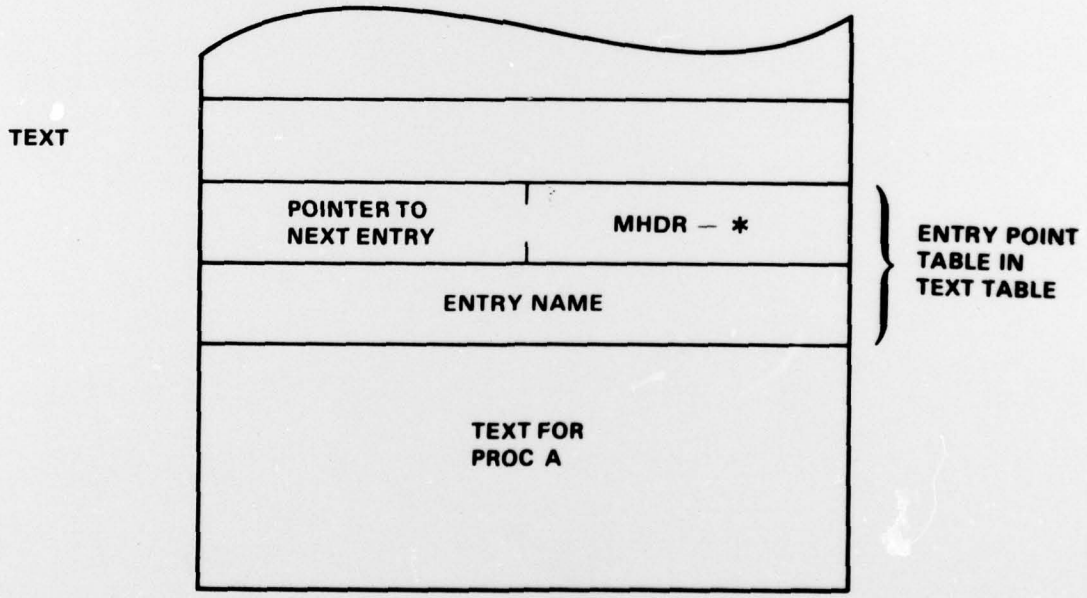


Figure 7. Text Table Containing an Entry Point Table

Description

This area of the object module contains the EASY machine language code for the program. It also may contain entry tables if there are entry points for PROC/FUNC in the module.

End Table (ENDM)

Purpose

1. To identify the end of a module

2. To function as a dummy-entry link into a module

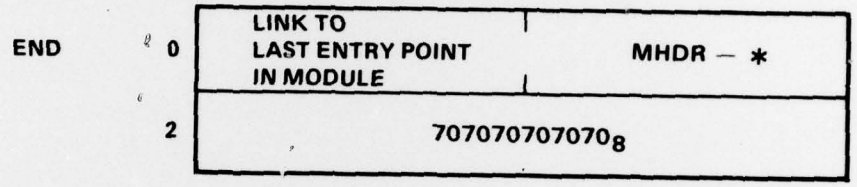


Figure 8. End Table

Description of Entries

Word 0 - Relative pointer to last entry point in the module

1 - Relative pointer to MHDR

2 & 3 - 707070707070 Octal. Dummy name for pseudo entry point in module

Note

Entry links point toward low memory, and external links point in the other direction. This choice was made so that the most often referenced entry points would be found the quickest.

LINKING ALGORITHM

During execution when a program addresses some procedure or data outside of its own module, a special microaddressing method is invoked. The machine instruction referencing the external does so indirectly through a two-word external table (see Figure 4). The machine instruction also causes the special microaddressing method to be invoked. This microaddressing method examines the external table address field for a non-zero value. If non-zero, it will use this value as the relative memory address for the memory reference or jump

instruction being executed. If the micro instruction finds a zero address value, the firmware will search all entry point tables (see Figure 6) for a name matching the external name. On finding a match it uses the memory location of the next address after the entry point name for the external address. This address is made relative to the external table and is written into the external table address field and the machine instruction proceeds.

If the micro instruction cannot find an entry point to match this name in memory and the external was a PROC or FUNC reference (CALL), then the loader can search its directories (libraries) for an entry point name matching the external name. If the external was a data reference, then the program is aborted. The reasoning is that the user should have data in memory before accessing it (this restriction may be debatable). Once found, the loader firmware loads this routine into memory, which writes (i.e., a block copy) the module directly into memory starting at the next available location by the BPS pointer. BPS is then set to be the first location of the module loaded. Therefore, the chain of external tables always starts at a fixed offset from the BPS and ends at TPS. Likewise, the chain of entry tables always starts in a fixed relation to the TPS and ends at BPS. Module loading in this manner allows the modules to be linked into the existing chains without any modification (e.g., relocation, resolution).

If the loader routine cannot resolve the external, then the name is unsatisfiable and the user program is aborted.* If the module has been found and loaded into memory, then the instruction which referenced the external is reexecuted. This time the micro instruction finds the entry point and normal execution continues.

As part of this load function the system routine (PSLOAD) must validate that the module is valid and there is enough field length between TPS and Top of Working Storage before loading programs. The PSLOAD function can be executed explicitly from the high-level language and load a module.

LIBRARY FUNCTIONS

The storage of routines on a file with a directory is referred to as a library. Loaders traditionally reference a file with a directory to place into core frequently accessed routines needed by other load modules. SORT and input/output (I/O) routines are examples of routines frequently needed by other modules. It is very desirable to have this type of routine accessible to other modules in a transparent (as possible) method.

MIL supports this feature by using the operating systems' file directory features. The names of the different files in a particular file library are the entry point names of the modules referenced. Thus the standard file mechanism of the operation system is converted into a list of entry points and the

---

* Unless the user has previously informed the operating system to return control with (or without) an informative message.

location of the modules containing them. The firmware uses the file directory mechanism to locate the module and then issues an I/O to transfer the required module into memory.

BOUND OBJECT CODE MODULES

Segment loading capability can be accomplished (as explained above) by making a call to an entry PROC/FUNC not residing in memory. Overlay loading can be accomplished with the aid of a simple copy utility which forms a single module of all modules common to one overlay. This overlay is then stored in the file directory for MIL with its primary entry point as file name. Any external reference to this entry point name will cause the entire overlay to be loaded. Execution of the overlay will start if it is a procedure entry point name.

DYNAMIC LOADING AND FREEING OF OBJECT MODULES

To load an overlay, a call is made to its external reference name. An entire overlay program space can be released by delinking any entry point in that overlay. Module delinking (freeing program space) is accomplished by the high-level statement PSFREE that translates into a micro instruction. The operand for this instruction is the address of any entry point in the module that is to be deallocated.

All modules with lower addresses than this module will also be deallocated at the same time. The micro algorithm uses the pointer which links to the beginning of a module to find the length of the module and therefore its last word address. Once this address is known, the algorthim then checks the stack used by SIMPL-Q. This is to insure that deallocation of the module(s) will not leave an activated stack entry for a module but no module to return control to. When this check is satisfied, pointers are adjusted for freeing of object modules. This is accomplished by setting the BPS to the location before the module being delinked. Also, the external address fields of the remaining external tables that reference the module just delinked are zeroed. This is performed by checking the addresses in the external chain against the new BPS. If the address is less than the current BPS, then the address is set to zero.

If the PSFREE request cannot be accomplished because of active modules with lower address than the current module requesting being freed, then the request is modified to deallocate as much of the program space as possible or the request becomes a no operation (NOOP).

CONCLUSIONS

The functions performed by the micro implemented loader allow for a wide range of capabilities without undue complexity. The design relies on a microprogrammable processor's ability to redefine the machine's apparent architecture and data structures to develop a relatively small comprehensible approach. The implementation has shown that this approach is easily produced, efficient, and adequate for the needs of the users.

REFERENCES

1. John G. Perry, Jr., *EASY System Programmer's Guide*, Naval Surface Weapons Center, Dahlgren Laboratory Technical Report NSWC/DL TR-3774, Dahlgren, Virginia, January 1978.

2. Charles W. Flink II, EASY--*The Design and Implementation of an Intermediate Language Machine*, Naval Surface Weapons Center, Dahlgren Laboratory Technical Report NSWC/DL TR-3765, Dahlgren, Virginia, December 1977.

3. V. R. Basili and A. J. Turner, *SIMPL-T: A Structured Programming Language*, U. of Maryland Computer Science Center, Computer Note CN-14.2, August 1975.

4. John Perry, *SIMPL-Q Reference Manual*, Naval Surface Weapons Center, Dahlgren Laboratory Technical Report NSWC/DL TR-3778, Dahlgren, Virginia.

5. D. W. Barron, *Assemblers and Loaders*, American Elsevier, New York, 1969.

6. C. Presser and J. R. White, *Linkers and Loaders*, ACM Computing Surveys, Volume 4 Number 3, September, 1972.

7. K. Dadl, L. Dadl, A. Mateeva, and I. M. Salamatin, *Organization of A Relocatable-Program Library for a Fixed-Page Computer, Computer Software and System Programming*, Plenum Publishing Corporation, New York, New York.

8. Roger J. Martin, *A General Purpose Overlay Loader for CDC 6000-Series Computers; Modification of the NASTRAN Linkage Editor*, Naval Ship Research and Development Center, NSRDC Report 9062, Bethesda, Maryland, April 1973.

9. Donald E. Knuth, *The Art of Computer Programming: Fundamental Algorithms*, Volume I, Addison-Wesley, 1969.

10. Burroughs Corporation, *Burroughs B7700 Systems Reference Manual*.

11. Nanodata Corporation, *QM-NCS, Preliminary Systems Operations Guide*, Williamsville, New York, May 1977.

12. Nanodata Corporation, *Task Control (TCP 1.05)*, Revision 2, Williamsville, New York, 1976.

DISTRIBUTION

Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, VA 22209
ATTN: William Carlson

U.S. Naval Electronic Systems Command
Washington, DC 20360
ATTN: John Machado
(Code 330)

Computer Science Department
University of Maryland
College Park, MD 20742
ATTN: Dr. Yahan Chu
Dr. Victor Basili

Computer Science Department
V.P.I. and State University
Blacksburg, VA 24060
ATTN: Dr. Richard Nance
Dr. Thomas Wesselkamper

Nanodata Corporation
2457 Wehrle Drive
Williamsville, NY 14221
ATTN: Mile Brenner

USC/Information Sciences Institute
4676 Admiralty Way
Marina Del Ray, CA 90291
ATTN: Lou Gallenson

Defense & Space Systems Group of TRW Inc.
One Space Park
Redondo Beach, CA 90278
ATTN: Barry Press
David Bixler
Herb Wagenheim

Martin Marietta
P.O. Box 179
Denver, CO 80201
ATTN: Skip Scown
B. Clausson

DISTRIBUTION (Continued)

McDonnell Douglas
5301 Bolsa Avenue
Huntington Beach, CA 92647
ATTN: Harris Dalrymple

RADC/ISCA
Griffiss Air Force Base
Rome, NY 13441
ATTN: Armand Vito

Dr. W. A. Burkhard
Computer Science Division
Department of Applied Physics and
Information Science
University of California at San Diego
LaJolla, CA 92093 (4)

Dr. John Tartar
Department of Computer Science
University of Alberta
Edmonton, Alberta
Canada T6G 2E1

Commander
Naval Ocean Systems Center
271 Catalina Boulevard
San Diego, CA 92152
ATTN: Code 5200
Russ Eyers

Commanding Officer
U.S. Army Harry Diamond Laboratories
2800 Powder Mill Road
Adelphi, MD 20783
ATTN: Branch 520
Rick Johnson

DIT-MCO International
5612 Brighton Terrace
Kansas City, MO 64130
ATTN: J. L. Herbsman

NCR E&M -SD
16550 W. Bernardo Dr.
San Diego, CA 92127
ATTN: Leslie Stevens
Manager, Product Firmware

DISTRIBUTION (Continued)

Nanodata
6065 Madra Ave.
San Diego, CA 92120
ATTN: Robert C. Boe

A. Ammerman (COMPRO)
Rt. 1 Box 690
King George, VA 22485

General Research Corporation
307 Wynn Drive
Huntsville, AL 35807
ATTN: H. D. Fitzgibbon

System Development Corporation
4810 Bradford Blvd, N.W.
Huntsville, AL 35805
ATTN: Robert Kirk

University of South Louisiana
Box 44850
Lafayette, LA 70504
ATTN: Paul A. Boudreaux

Defense Documentation Center
Cameron Station
Alexandria, VA 22314 (12)

Library of Congress
Washington, DC 20540
ATTN: Gift and Exchange
Division (4)

Local:

E41
K54 (R. Pollock, W. McCoy) (2)
K60
K61
K70
K74
K74 (Gass) (20)
N30 (R. Hein)
X2101 (GIDEP)
X2102 (2)